ASYNCHRONOUS DATA EXCHANGE LAYER BETWEEN A MOBILE DEVICE AND SERVER

Adam Krupski

Bachelor's Thesis

May 2013

Degree program in Software Engineering

Technology, communication and transport

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s):	Type of publication:	Date:
KRUPSKI Adam	Bachelor's thesis	29.05.2013
	Pages:	Language:
	43	English
	Confidential:	Permission for web publications:
	( ) Until	(x)

Title:

ASYNCHRONOUS DATA EXCHANGE LAYER BETWEEN A MOBILE DEVICE AND SERVER

Degree Programme:

Software Engineering

Tutor(s):

Lappalainen-Kajan Tarja

Assigned by:

Nestronite Oy

Abstract:

The objective was to design and implement data exchange module cooperating with Android operational system. The module itself should provide a queue of data to upload or download and should be resistant to the connection drops. Data exchange should take place in the background without user interaction. Immediately after receiving the data it should be approved and available in the application.

The module was first designed, implemented and tested using an example application. The next step was to unite with the company's projects and adapt it to existing database. The last task was to build in a module to the second project, slightly different in the structure of the operational data. The main problem was to design a flexible and reusable data exchange layer. To achieve the set objectives, the supported information was contained in the documentation for the Android operating system.

The thesis explained what Android operating system is and how it works. It also describes what the HTTP protocol is and how it can be used for data exchange. Finally, the method of the two data exchange protocols is described. The module is used by the companie's projects and will be further developed.

Keywords:

Mobile device, Android, REST, cloud file storage, data transfer, HTTP/1.1

Miscellaneous:

TABLE OF CONTENTS

Figures

List of acronyms

- API - application programming interface used to determine the structure of request sent to the server. Also used to determine the available methods provided by the library.
- App - Often used abbreviation by developers and users for the word "application"
- Apple iOS - The operating system used by Apple on smart-phone devices. The biggest competitor of Android in the smart-phone market
- BitTorrent – protocol designed to exchange data between distributed users. User's computer download and upload data even files is not fully downloaded.
- GUI – Graphical interface to allow user to interact with electronic device using images broadly defined multimedia.
- MIME - Multipurpose Internet Mail Extensions, used to support header and body in: non ASCII catharses, non text attachments and multiple parts of body. Used as extension to email format, also extension to HTTP protocol.

1 Introduction

1.1 Background

Mobile devices are commonly used by people around the world, right now cell phones are something more than just a device to make phone calls and receive messages with. In time they have become faster, using more features and hardware. Smart-phones have become easy to use, small devices with easy access to Internet. This has resulted in wide interest of creating various applications to help our lives by hobbyists and also by the business sector. Couple of companies have gone into this business and provided their own operating system. Today only iOS, Windows Phone, Symbian, BlackBerry and Android have influence on this specific market. The Android operating system with saless bigger than the whole other competition put together.

Applications have capabilities to connect with servers and download or upload data, opening brand new features for mobile devices. This data exchange process has become a complex problem on the device from which the user requires fast operation and smoothness of flow. Exchanging big amount of data between mobile device and server has become bit harder to solve when device can lost network connections or narrow bandwidth. This leads to outwardly simple problem of exchange data more interesting and complex.

1.2 Android System

Android inc. was founded in 2003 initially to build operating systems for digital cameras. After a short time, it turned out that the camera market by itself was too small. The company changed their target devices to smart-phones , this time to rival with such competitors as Symbian and Windows Mobile. The name of the operating

system was chosen to be the same as the company name, Android. Google Inc. acquired Android Inc. in August 2005, employing all the founders and employees. (Wikipedia, Android, 2013)

In November 2007 Google started a consortium called the Open Handset Alliance (OHA) with 34 members. Right now the consortium has 84 members. The objective of the consortium is to develop open standards for mobile devices. OHA is composed of application developers, hardware manufacturers, mobile handset makers and some mobile carriers. The main product of the consortium is the Android system, released under an open-source Android license. (Wikipedia, Android, 2013)

The first device utilizing the Android operating system was sold in October 2008 and in a short time Android became very popular. At the time of writing over 69% new smart-phones devices use Android as their chosen operating system. The popularity and flexibility of the system has resulted in an extension in the list of supported devices, such as televisions, game consoles, smart-watches, cameras, and notebooks. This rapid adoption curve is illustrated in Figure 1. (Wikipedia, Android, 2013)

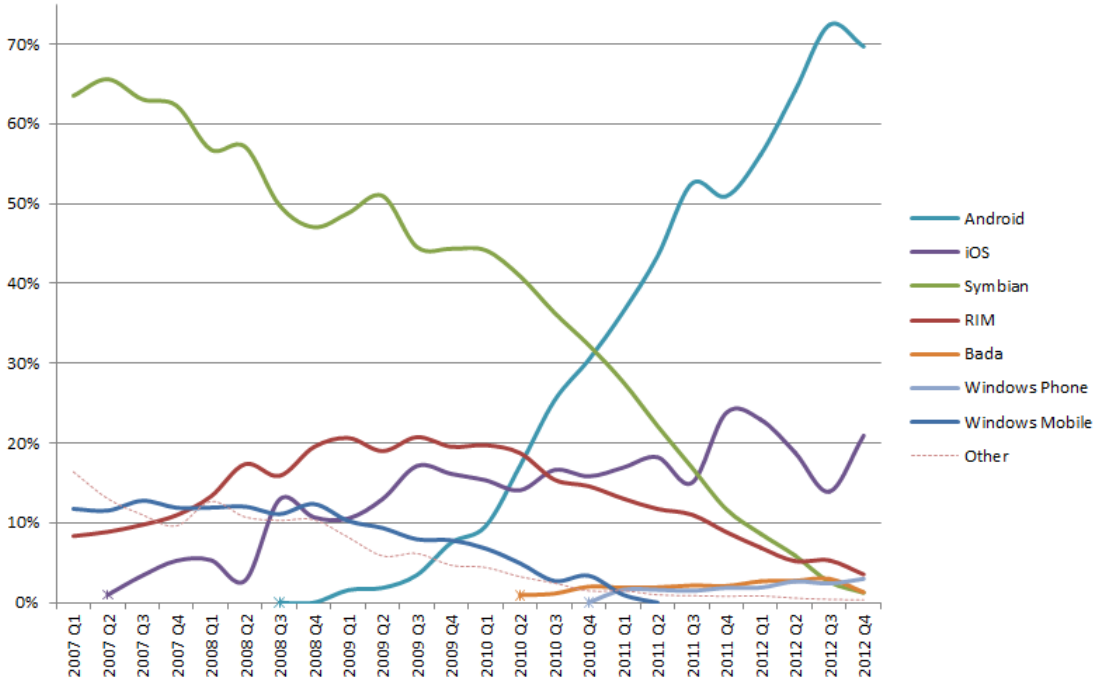

Figure 1 - World-Wide Smart-phone sales in percentage scale (Gartner, 2013)

1.3 Android Application storage

Google Play is the main service for distributing free and paid applications for the Android operating system. It was first opened to users in October 2008. Four months later from this initial opening support for paid applications was introduced in The United Stated and United Kingdom. Unfortunately, to this day not every country supports selling applications for money yet. Right now 32 countries are supported; information which countries are supported can be found on Google's support site. Installing applications via a desktop browser was introduced in February 2011. Users that access the store by using an account bound with their device, can select applications to install and pay for them on their computers. The selected applications will be installed on device automatically. (Wikipedia, Google Play, 2013)

In March 2011 Google introduced in-app billing, using this extension of functionality users can purchase artifacts and items straight in their applications without going to the Google store app first. In July 2012 Google redesigned their market interface and extended the service to also sell books and rent movies. After two months from this milestone users could also buy music via Google Play. (Wikipedia, Google Play, 2013)

Based on a Google report, the service currently maintains over 800 000 applications. To buy an application, users need to create a Google account and bind it to their device. A user is able to change their device to a new one without losing already paid applications, for this reason Google stores information about an acquired application on account. Users can just take a new device into use and bind it to their account. The applications from the old device will be installed automatically; also contacts will be synchronized with the account. (Wikipedia, Google Play, 2013)

To upload a new application to Google Play, developers or companies need to pay 25$ to create account. Of course, many applications can be sold from one account. Right now the maximum allowed application size on Google Play is 50MB plus two extra files 2GB each, (4146MB of application including extra files). Before sending an application to market the application needs to have a digital signature. (Wikipedia, Google Play, 2013)

Every single application is digitally signed by their authors, also during the development process an application is signed automatically by the Android development tools using a so called “debug key”. An application cannot be published on the Android market without using this key. Additionally, the key expires after 356 days. To publish an application on the market the developer (or company) needs to sign the application with their own unique key or a set of keys. When key validity time is expired, the user can use the application but an upgrade procedure will be complicated. Instructions on how to sign an application are available on the Android

Developer site in the section “develop”. To publish an application on Google Play, the date of expiration of the used key must be set to occur after 22 October 2033. Google Play enforces this date on authors to seamlessly upgrade current applications. (Wikipedia, Google Play, 2013)

1.4 Android development

To build and develop applications Google provides a Software Development Kit (SDK for short). The SDK is a bundle of tools to make the development process as smooth as possible. The set of tools is a grouped by the Application Programming Interface (API) Level. The API level is dependent on the Android system version (see figure 2). The tool “Android SDK Manager” provided by Google is a program to download actual versions of tools and files such as (Android developer, Android SDK, 2013) :

- The SDK platform – this tool is required to build an application and it contains the API itself and required libraries.
- Emulator – This is a program meant to emulate one of the devices. Developers do not need a physical device to develop an application, although not every operation that can occur on a cell phones possible to emulate, for example incoming phone calls.
- Application samples.
- Source code of the SDK.

Right now the newest version of the API level is 17 (code named Jelly Bean). Google provides encompassing backward compatibility between old and new versions of the API, which is one of the great advantages of the system. However to choose which one should be used for developing a new application is always not so obvious. Over time, many devices have been sold with many different Android system versions and API levels. To estimate how many devices use a specific API level, Google publishes reports about device usage details such as system versions. The data is collected every two weeks and is published on the Google Android web site. (Android developer, Dashboards, 2013)

TABLE 1 – relative numbers of working devices on given API level. (Android developer, Dashboards, 2013)

Version	Codename	API	Distribution
1.6	Donut	4	0.1%
2.1	Eclair	7	1.7%
2.2	Froyo	8	4.0%
2.3 - 2.3.2	Gingerbread	9	0.1%
2.3.3 - 2.3.7		10	39.7%
3.2	Honeycomb	13	0.2%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	29.3%
4.1.x	Jelly Bean	16	23.0%
4.2.x		17	2.0%

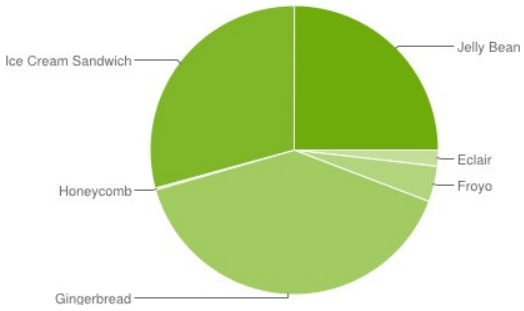

Figure 2 - Android platform versions (Android developer, Dashboards, 2013)

The decision which API level to use is related to the amount of devices on the market, and how many devices can run the application. The largest collection of devices currently use the API level 10 because many devices were and are, at the time of writing, sold with this system. Right now over 94% of the devices run an equal or greater version of the API at level 10. The devices with a relative low API level are no longer produced and will be regularly phased out in time.

1.5 Android application architecture

Android architecture is divided into 4 basic layers. This construction gives clear a vision on the system and an easy view how deep applications are integrated in to the system. To develop for each layer developers need to use other tools provided by Google. The following figure illustrates how the system components grouped into layers. (Android developer, App Framework, 2013)

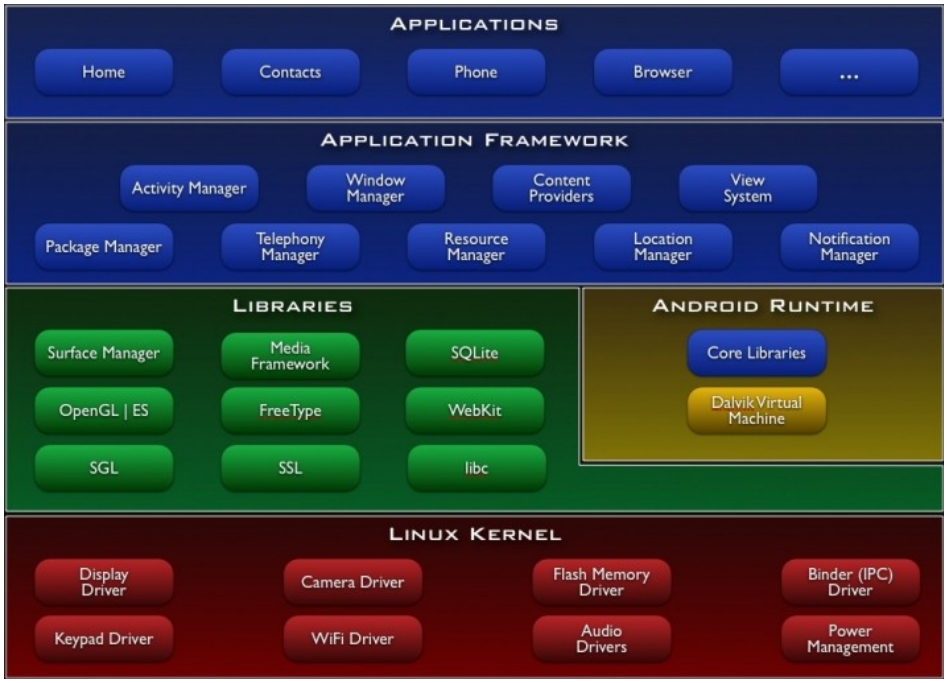

Figure 3 - Android architecture diagram (Android developer, App Framework, 2013)

1.5.1 Applications

The highest levels of system cake, the most of the application are integrated into just this layer. Android is designed to be used primarily in smart phones, so the architecture also includes some basic phone applications delivered with the system. This application makes the device useful, a phone without a “Dialer” (Android default application to make a phone call) or a web browser would be useless. This set of applications shares basic data with other applications, for example the contact list. Applications can be installed on a devices internal memory or external memory (most commonly a SD card). However, every single base application needs to be installed on the internal memory.

An user of a smart-phone cannot remove these pre-installed applications. However, he or she can install a new one with similar functionalities and in device configurations set the new app to be used as the default app for the action. (Android developer, App Framework, 2013)

1.5.2 Android framework

The second level below applications is full of managers. This layer is responsible for keeping an application running and providing them with the data they request. One of the most important managers is Activity manager for keeping a running activity along the special live cycle (see a more detailed description in the Activity section). Android tries to keep as many programs as possible running until the exhaustion of memory or another resource. This layer decides which program will be removed and restored based on hardware usage. Android framework also provides access to data stored in the phone or external memory and device data pipe as well (for example, GPS status). (Android developer, App Framework, 2013)

1.5.3 Libraries (Android Runtime)

Almost the whole layer is written in C or C++. The code written in this language does not need to run its own virtual machine instance like Java resulting in fast start and execution of code. Most of the libraries used in this layer that originated in open-source projects are tuned by Google to be efficient in mobile devices. This layer provides features like: SQLite (a database engine), 3D libraries (based on OpenGL ES 1.0), media libraries, LibWebCore (browser engine based on Apple's webkit) and several more. (Android developer, App Framework, 2013)

Developers can use this layer from the source code level. They can also add new own libraries, that are called “native libraries”. The native code is written in C or C++ to provide fast execution in situations where results should be ready almost in real time. An example of such a need would be for example simulation of physics.

In addition this level provides Java core libraries and Dalvik Virtual machine instantiates. Dalvik is an open-source Java Virtual machine responsible for running the application on the device. It is optimized to have a low memory requirement, to support multiple instances of the VM, memory management and thread support. (Android developer, App Framework, 2013)

1.5.4 Linux kernel

The most important and lowest part of the system is the kernel. The kernel is a program to manage hardware and device resources. Android kernel is not exactly the same as Linux kernel, being a modified copy called a 'fork' maintained separately by Google. The maintainer of the kernel cut off non-used parts of the GNU/Linux -operating system's kernel (called 'Linux'). The device manufacturers also take part in kernel development and maintaining. Every manufacturer of smart phones (or other devices) uses their own hardware and this equipment needs to have device-specific drivers that can cooperate with the Android kernel. In addition, the kernel is responsible for security, network, process management (excluding Activities) on a low level. The Android kernel in system versions under and equal 3.2 is based on Linux 2.6, above this the version of the kernel is based on Linux 3.0.. (Android developer, App Framework, 2013)

1.6 Android Applications

Android applications are written in a version of the Java programming language (slightly altered by Google). Compilation of the source code into working binaries has two steps. First the provided Java code is compiled by a standard-compliant version of Java Standard Edition compiler to byte code, an intermediately language. The second step of the process is convert this byte code to Dalvik byte code and save this resulting file with a “.dex” file extension. The compilation process is complete but all other related files are still left unstructured. Android developer tool then generates a package with an “.apk” file extension which contains only one Android application. The “apk” file is nothing else than a compressed folder containing all executable files (.dex), the application's resource files (like images, music, databases, etc), certifications and a special Manifest file using ZIP file format. This package format is similar to JAR structures used in desktop versions of Java. (Android developer, Application, 2013)

The Manifest file contains information about Java classes used by the application, all privileges from the system required to run the application, and finally it lists required

hardware (for example an application to capture image do not have any function on devices without a camera equipped). (Android developer, Application, 2013)

The creators of Android want to create a very secure system. For this reason, every single application has their own virtual machine instances and run in isolated sand-boxed environments. Android provides some basic blocks to create an application simpler and faster. Each component has a different role in the application to solve different issues. There are 4 components that can be used by developer closed in classes. This 'Glue' to communicate or create them is called the Intent object. (Android developer, Application, 2013). Intent is an abstract object that works like a message addressed to a specific system or to applications. An intent object can start new Activities, start background Services (or just deliver messages they already exist), and send events to Broadcast Receiver. This object represents passive abstract data and the instructions of actions to execute. (Android developer, Intents, 2013)

1.6.2 Activity

Activity is the basic block to building an application on Android and represents the graphical UI to user. Every single application uses this block. Because many issues can occur on a device (for example a new phone call), the activity has a special, unusual live cycle to pause, hide and be replaced by another activity of another application. Figure 4 shows the key points in the life of any given activity. (Android developer, Activities, 2013)

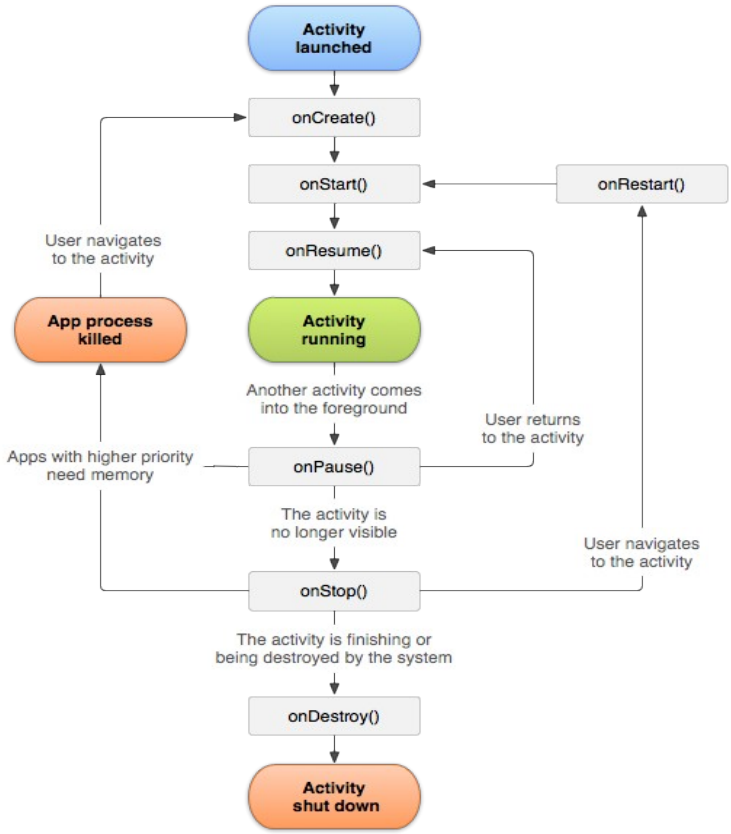

Figure 4 - Android Activity live cycle (Android developer, Activities, 2013)

Short description of every key method is as follows:

- OnCreate – this method is called when an Activity needs to be shown and the object does not yet exist in the system memory. This is usually used to set a layout for the Activity and to fill the layout object using some default data.
- OnRestart – can be called only after the onPause event was called. Only this method can determine if the Activity is created or is restored from the background memory.
- OnStart – this method is called only when the application returns from the background to foreground. Also this method takes part in the first creation of the Activity.

- OnResume – the final stage of running the application. Also this method is called if any pop-over window disappears and this Activity has control on the full screen.
- OnPause – is run only when Activity GUI interacts with lightweight message boxes created by the application, for example confirmation dialogs.
- OnStop – is called when the Activity is hidden by another activity being woken up. This method also is called when another activity is started from the same application.
- OnDestroy – method will be called when the system decides to destroy the Activity, and all resources to maintain the Activity will be released. Since this time user can not navigate to this Activity, the Activity can only be show again when it goes through the whole creation process.

A common mistake done by developers is handling Activities similarly to Linux processes. Even if all Activities are closed and removed from system, their background process can still exist, for example a handled Service in the background. (Android developer, Activities, 2013)

1.6.3 Service

The service can be a part of an application running in background and executing code using an autonomous thread not limited by the life cycle of the main application's Activities . With this solution, the main thread can work without waiting for results from the service and only get said result when it is ready. The Android SDK provides functionality for two differed types of services: “Started” and “Bound”.

To create a “Started” service, one of the components in the main application need to call a method called “startService()”. By default a new service will be started if an instance of it does not exist, otherwise a new intent object will be received by an already existing service. Once started the service will be executed forever even when user close whole an application. Developers need to remember to stop this type of service manually. Usually a service does not interact with the GUI and user, so it is

the right place to put a queue of tasks to be executed without waiting for the task's results.

The “Bound” service works in a client - server style of communication. To create this type of service, developers need to call the “bindService()” method. This function needs to return an interface to communicate with the client component. If the method returns a “null” reference, then a service cannot be bound and work identically to a “Started” service. Developers do not have to worry about properly stopping this kind of service. When all clients unbind from it, the service will be automatic stopped. (Android developer, services, 2013)

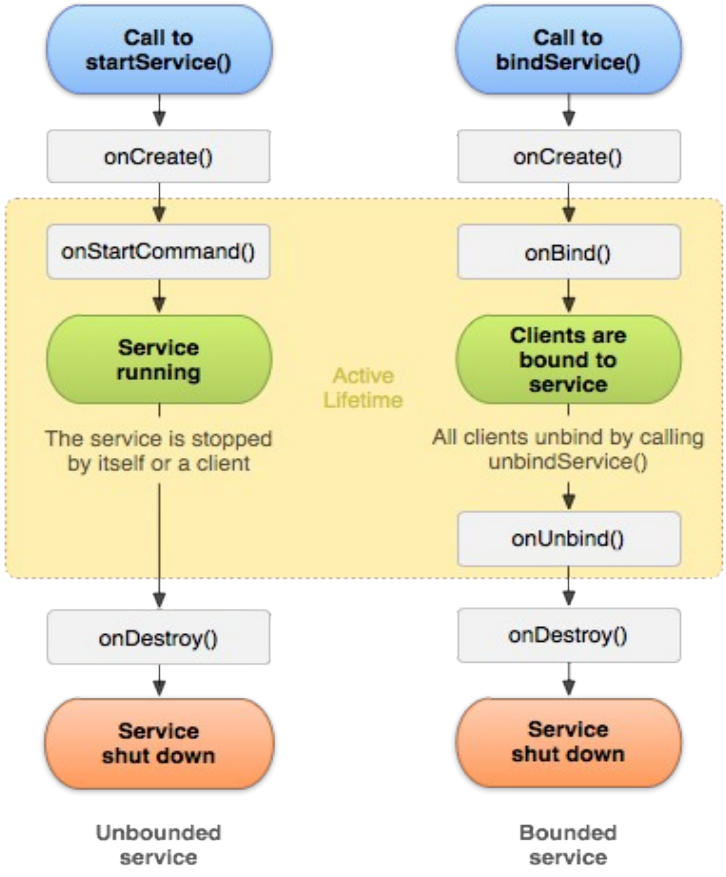

Figure 5 - Service live cycle. Left site show “Started” service, Right site show “Bound” service type (Android developer, services, 2013)

Similarly to Activity life cycle methods, “onCreate()” is called when a service is created and “onDestroy()” is the last method in the service process's life cycle.

The Android system tries to keep all services running at the same time. When the system needs more hardware resources (for example memory) to more current operations, services will be destroy by the system and removed from the process list. Developers can set service attributes to run it again on termination, when more free resources are available. This makes the service a very powerful tool for an application and easy to manage. (Android developer, services, 2013)

1.6.4 Broadcast Receiver

Broadcast receiver works like triggers when system sends any events, for example about low battery level. Functionality of Broadcast Receiver is invaluable when the application needs to be activated after some event. By default broadcast receiver is activated in every system event, for this reason developers can set filters in the Manifest file and run receiver only if an event is matched to this filter. Not only can the system send broadcast messages, also every single application in the system can do the same. (Android developer, Broadcast receiver, 2013)

1.6.5 Content provider

Content provider represents mechanisms to manage application data, basically databases and files. This block is bit more complex that other blocks, mainly because developers also need to design all data privileges in line with the functionality. Using this block, developers can be specify which data will be shared with group of extended applications. In addition special allowed methods can be specified (read, write or update data). Application without shared data functionality does not need to implement this block and can use default one automatically implemented by the Android SDK.(Android developer, Content providers, 2013)

1.7 Online technologies

1.7.1 HTTP/1.1

Hypertext Transfer Protocol (HTTP) is a high level protocol that uses a request – response architecture. In 1991 the protocol version HTTP v0.9 was released, and it was used only by the World Wide Web (WWW) to exchange data between browsers and servers. At this time every single request was started by a “GET”, as it was the first group of methods in HTTP. Every single request references to “Uniform Resource Identifier” (URI) which uniquely identifies a wanted resource. (RFC 2616, 2013)

The GET method is nothing more than a request to obtain some wanted resources from a server. The protocol was not perfect in the first version, the response containing only HTML and ASCII characters. In 1996 the HTTP v1.0 version of the protocol was finished and released. A range of methods were extended and introduced, like HEAD and POST. In this version the Multipurpose Internet Mail Extensions (MIME) methods were introduced with small modifications from the original proposal. By using MIME, HTTP could send data with non-ASCII coding and even binary data (for example images and files). (RFC 2616, 2013)

The HEAD method is request only meant to query the server for meta data about the resources that will be loaded. This initial query allows client applications to get basic information about the resources to be loaded, for example last modification date (invaluable information to any browser's caching system). The POST method is a request that allows the client to send and save data on the receiving server. Using this method user are allowed to send for example input from contents or even files. In additions the PUT, DELETE, LINK and UNLINK methods technically are a part of the HTTP specification as an appendix, not used widely. The current version that is in use, HTTP/1.1, was released in 1997 and then appended with an upgrade 2 years later. Available methods for usage are GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS and CONNECT. The PUT method is used to update data residing on the

server or create said data if it does not yet exist, consequently the DELETE function can be used to remove data. The OPTIONS method is used to get information by a client about possible communication options with the server, for example the used compression and encryption methods. The TRACE method is used usually to test indicated server, by asking it to send a request as a reply to the TRACE. Finally, the CONNECT method is used exclusively by proxy servers. (RFC 2616, 2013)

Every single HTTP response has a 3 digit “reply code” always appended at the beginning of header of a response, divided to 5 different groups:

- 100 – 199 – Informational Codes
- 200 – 299 – Successful Requests
- 300 – 399 – Redirection Codes
- 400 – 499 – Client error Codes
- 500 – 599 – Server error Codes

1.7.2 REST

Representational State Transfer (REST) is a lightweight, distributed model of designing Web-based APIs. REST is used to store, update, remove and grab data form a Web Service that implements the RESTFUL design philosophy. REST was originally defined in 2000 by Roy Fielding, one of the designers of HTTP/1.0 and HTTP/1.1 protocols. In that time REST was a parallel project to HTTP/1.1 and was based on HTTP/1.0. This technology uses a request – response style of communication. The REST protocol does not provide sessions. REST uses only 4 methods of the HTTP/1.1 pool to manage data, which may be related to a set or a single data:

TABLE 2 Available methods offered by the REST service

Method	Set of data in request	Single data in request
GET	Get all sets of data	Get one item of data
POST	Create many items	Not used or used to create new fields in the specified item
PUT	Replace given items	Replace item or create it if not exist
DELETE	Delete set of data	Delete one item

This set of operations gives ability to easily manage data.

On the first glance it looks like there is one difference between REST and a more classic database interface. However in REST requests for data not only fetches the desired items from the database, a REST server can also generate some data to its reply or fetch items from other servers. JSON files are commonly used to represent the resulting set of items. (REST, 2013)

1.7.3 JavaScript

JavaScript is a high level, interpreted script-based programming language originally known as ECMAScript. The first version of JavaScript was released in 1995, and built bundled in the Netscape Navigator 2.0 beta. This first version was named LiveScript. It was renamed to JavaScript when the Netscape 2.0 Beta 3 was released. JavaScript was designed to support both server and client applications, but currently most of the applications in JavaScript are written on the client side. The main reason why JavaScript is so popular nowadays is because of the possibility to execute custom code through the browser. The HTTP protocol can only send responses and this data will then be interpreted by the user's browser. JavaScript van also send a response message and browser run this code using one of its script engines. JavaScript is an interpreted language and needs a browser-based script engine to use it. Right now almost every browser has their proprietary ECMAScript engine. Some of these are under open-source licenses, like Google's V8.

Node.js is the most popular server side JavaScript engine. In terms of performance, Node.js is fast which has contributed recently to increase the popularity of server-side applications written in JavaScript. (Wikipedia, JavaScript, 2013)

1.7.4 Node.js

The project was founded in 2009 by Ryan Dahl. The main goal was to create a web site with pushing data functionality. Node.js uses Google's open-source V8 JavaScript engine and it is meant for creating servers using this script programming language. The Node.js software does not need any extra web servers to function, having the ability to create a fully functional server by itself. The Node.js project model is based on events and on a non-blocking architecture. Node.js has become a good and robust for creating web services because of these features. (Node.js, 2013)

1.7.5 File upload

Uploading files from mobile devices is not so obvious as it may initially sound. Multimedia files that need to be sent are usually big in terms of size and the network connection for the device may not be stable. Many events may occur during this upload event, such as the network connection disappearing, a new phone call may be arriving, even low battery power or many other issues. The application developer needs to take care of all these events, while taking only the least CPU time as is possible. If an application consumes too much CPU power it can reflect directly to the devices battery life and cost energy, when in fact the device should work as long as is possible.

1.7.6 Amazon Simple Storage Service (S3)

Simple Storage Service, in short S3, is the online storage web service founded by Amazon. First it was introduced in March 2006 in the United States and in November 2007 in Europe. It is considered the first public web platform as a service. S3 provides

REST, SOAP and BitTorrent interfaces to send content to the S3 cloud and basic management capabilities of accounts (except for the BitTorrent protocol). (Amazon AWS, 2013) At the time of writing the price for 1GB of data storage is less that 10 U.S. cents per month, with an additional request cost of less or equal (depending on the request being made) of 0,005$ per 1000 requests. Transferring data out is free, but sending data back to the service costs less that 0.120$ per 1GB uploaded. A full table of costs incurred can be found on the official S3 web page. In June 2012 Amazon reported information that over a trillion objects are stored in the service. Amazon guarantees 99.9% monthly up time. (Amazon pricing, 2013)

Amazon provides storage folders called buckets to store files and objects. Bucket functions like a container for the files. A system or company administrator can manage privileges on the whole bucket and can also set the bucket attributes to be displayed as public. If a bucket is set public, every object contained within is granted a URI. By using this URI any client can access the file for example by using a browser. (Amazon AWS, 2013)

2. Problem

The main purpose of the proposed application is to get feedback about products or services by using a mobile device that saves the files on a remote server. The most important backbone of the application is capturing pictures, video and audio files about a product or service. Then save files on the device and consequently uploading them to a pre-specified remote server. From the server the files are tunneled to the Amazon S3 file cloud.

Application should use two different technologies to send files, first REST to send files to the backend and Amazon S3 from the backend to the cloud. In addition the application should consume the least amount of device resources as is possible for maximum battery life.

2.1 Research question

The challenge in this thesis is designing and implementing a mobile application layer for an Android application to allow saving, exchanging and syncing data and files between the local device's storage and a central server infrastructure connected to a file-storage cloud. The complex main question that was to be answered was: “What should the syncing service work flow look like?” Additionally, this main question begs the follow-up queries:

- Should the service use a queue of threads that wait for free network connections or should it build one big queue of files to send?
- Should the service be working in the background the whole time, waiting for new files to send, or should it be activated only when necessary?
- Where to write meta-information about files that are to be sent?

The second major problem of asynchronic data exchange was “Which moments to trigger the updating method for downloading new content is the best?” Of course new data should be download when it appears on a backend, but how exactly does the device notice this new information?

2.2 Proposed architecture

The first approach to solve this problem was as follows.

The application should send data in a non-blocking section of code. The initial idea to solve problem was thought to be to create a service for managing syncing threads. When some artifacts successfully captured, an Intent object should have to be sent to start the syncing service. Information about path of files should be sent as extra data. The service should create one synchronization thread for each file and put them in a queue of threads and execute them one by one in order. The sending feature should use Amazon S3 REST API to create an active connection, create a bucket if one does not exist and send the file. Captured file should 3 possible statuses saved in the frontend database:

- new file – file has been created and is ready to be sent
- sent – file was successfully sent
- removed – file was deleted because of some reason, for example user changed external storage memory cards.

All to captured artifacts are obtained by using default Android applications.

The backend is split in to two services. The project must offer a very secure and stable storage of files, on the other hand it must be easy to manage, flexible and fast in changing contents of data offered to and from the backend. Amazon S3 provides secure and stable storage of files. A REST service provides data, easy access to administrative functions and is easy to change. Using Node.js and JavaScript language, the idea was to create a REST server to provide synchronizable content. Downloaded files should be easily available, parsed on the device and then saved into a database. This part provides only text-based JSON structured data. Amazon S3 file storage should be used to store all artifacts captured by user.

2.3 Development methods

To develop this project the Eclipse software development environment using the Android Development Tools (ADT) plugin was chosen. Google provides a very powerful plugin to Eclipse, an interface of which is simple to use. Android developer tools induce many programs like: an emulator, a debugger, a key manager, etc. It is possible to use these tools without eclipse; however using them in such a way is more time-consuming. The ADT extends the Eclipse IDE with Android specific debuggers and a virtual device emulator for fast testing of code.

To communicate with another members of develop team and to send parts of code, Git was used. Git is a distributed version control system. This system saves a lot of time and pain to share code with another team member.

2.4 Tools

Git – is a distributed revision control system and source code management tool. Development of Git started in April 2005. The release of version 1.0 was on December 2005. It is very popular system to manage big and small code projects. One example of a big open-source project managed with git is the Linux kernel. (Git, 2013)

Tcpdump – is an open-source tool for capturing network packages. It is a very useful tool for seeing how exactly the whole outgoing and incoming packages (for example HTTP packages) look like. During development this tool helped to investigate which side of the system sent wrong requests.

Eclipse – is a development environment that works with many programming languages. It uses a custom own Eclipse Public License but in general can be thought to be freeware and open-source. Eclipse was founded in 2001 by a consortium of companies which includes: IBM, Borland, Red Hat and a few more. In January 2004 the Eclipse Foundation was created and took over the Eclipse IDE project. The

software does not provide any tool to support programming language in itself. Eclipse has an extensive system of plug-ins and due to this flexibly adapts to almost any modern programming language. (Eclipse, 2013)

Android device emulator – A program provided by the Android SDK, downloaded from the Android developer web page. This program emulates one a physical Android device.

3. Development

The first iteration of the synchronization technology was designed to send data to Amazon S3 by using an external library straight from the application itself. The first iteration started even before a full design session. The main purpose of the development project before designing was to learn the Android development style and possibilities. The AWS S3 library also was a new technology to use.
To exchange data with the Amazon S3 service, a developer needs to use a custom Amazon AWS library for Android. This library is enclosed in a JAR file and can be added to the eclipse project easily. The AWS library gives developers the possibility to manage S3 data and to send such data to the S3. Similar functionality also is available at Amazon develop web site.

Android does not have any built-in architecture library to use multipart HTTP request. The multipart request provider sends one HTTP request in many parts. Of course, a small request does not need to use this technique. However, the request type saves a huge size file with some metadata, for example a high resolution picture with a description, and thus allows sending the file request in parts. With this technique it is possible to lose the connection between device and server. If the connection is restored then the device needs only to send the missing parts, not the whole the request. Httpmime is a Java library provide by non-profit Apache Software Foundation to support MIME multipart encoded entities. This library allows to easily build HTTP request. Developer has possibility to change or add any fields in header.

3.1 Initial period

The second iteration of the sending technology was by using a custom-made RESTFUL web service and Amazon S3 web service. Amazon S3 does not include plain-text data exchange; for this reason, the two technologies were implemented. Amazon Web Services is very stable, since sending files is critical in this application determined to use this solution. To solve compatibility problems of these two technologies a “Class Adapter pattern” was deemed to be useful. This pattern solves

the problem of fast switching between many technologies which do essentially the same work in a single project. The adapter pattern divides the problem to several classes and interfaces. A client class contains a reference to an executor. The executor implements methods of an adapter. One of the executors uses Amazon and the second REST technologies. The choice of which executor should be used is set in a configuration class, in a static method called getSenderExecutor, that gives references to the class with the correct technologies. Additional benefits include the fact that information is saved in one place and executors are easy to change. The flow diagram 6 shows the adapter pattern for the thesis project in UML notation:

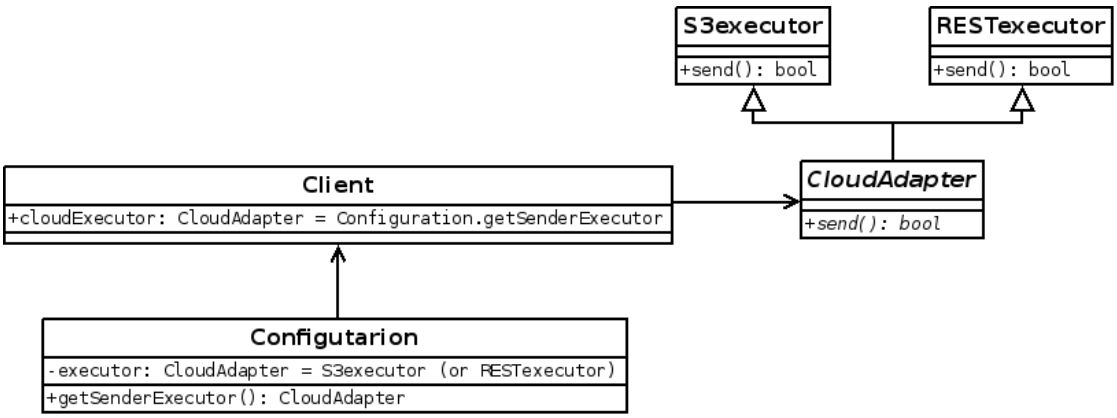

Figure 6 - UML diagram of adapter design pattern modified to the project purpose

Database

The designed module needed to be asynchronous, meaning for example in the case of a lost network connection data will be synchronized when the connection resumes. Files to be sent are stored in either external (memory card) or internal memory. To store data represented as plain text the internal Android library database SQLite was used. This library provide data storage as SQLite file notation, in a application's private file space.

Android provides a SQLiteOpenHelper abstract class which can be extended to an application's own purpose, can be used to easily create and manage versions of a SQLite database. Methods onCreate and onUpdate need to be implemented when extending this class. The method onCreate is called only once when the application

first run. The useful method onUpdate is called only when the application is upgraded and when in such a situation the database also needs to be updated. This method takes 2 important arguments: the old and new versions of the database being updated. The database manager calls this method if it is needed.

3.2 Reformulating application

Whole communication was moved to our custom-made service, also exchanging files with S3 was desinged to go through our service. This solution solved the problem of storing Amazon S3 credentials, that created a risk of reading credentials by third parties, eliminating it. The solution increased the flexibility of the backend service in case of attacks by malicious users or any other malicious parties. Expansion of the service is possible at any time and direction; since the S3 is very stable and resistant to failure. Any attackers, based on their Android device-id can be put to a blacklist and every request from them can be dropped in a malicious case.

External server credentials

Java byte code is vulnerable to reverse engineering, every single application needs to solves major issue that is where to store required credentials for external services, for example for accessing Amazon S3. Amazon recommends several complex solutions for this problem; however, a decision was made to move the whole S3 communication to the REST-based backend server.

Capture Activities

After testing on multiple devices of varying API levels and model ranges, it turned out that some capture applications do not operate correctly. Because the user can change default capture applications, not every application works as is to be expected. For example, using one of the device's built-in picture capturing activity after taking a photo is ready to take another one (giving the application authors to have the opportunity to improve the photo). This logic problem resulted in the user not being able to accept a picture and being forced to press the hardware “back” button, sending the main application a wrong error code indicating failed capturing. To solve this

problem the application was developed further to include it's own capture activities, allowing for a better usability flow and for better management of resultant data.

3.3 First pilot

First version of application needed to download content from the backend server and present it to the user. Contents were nothing else like instructions to take photo or video of objects. This solution allows maintainer of the backend to have control over all the time at content of application (with the exception when device does not have access to network).

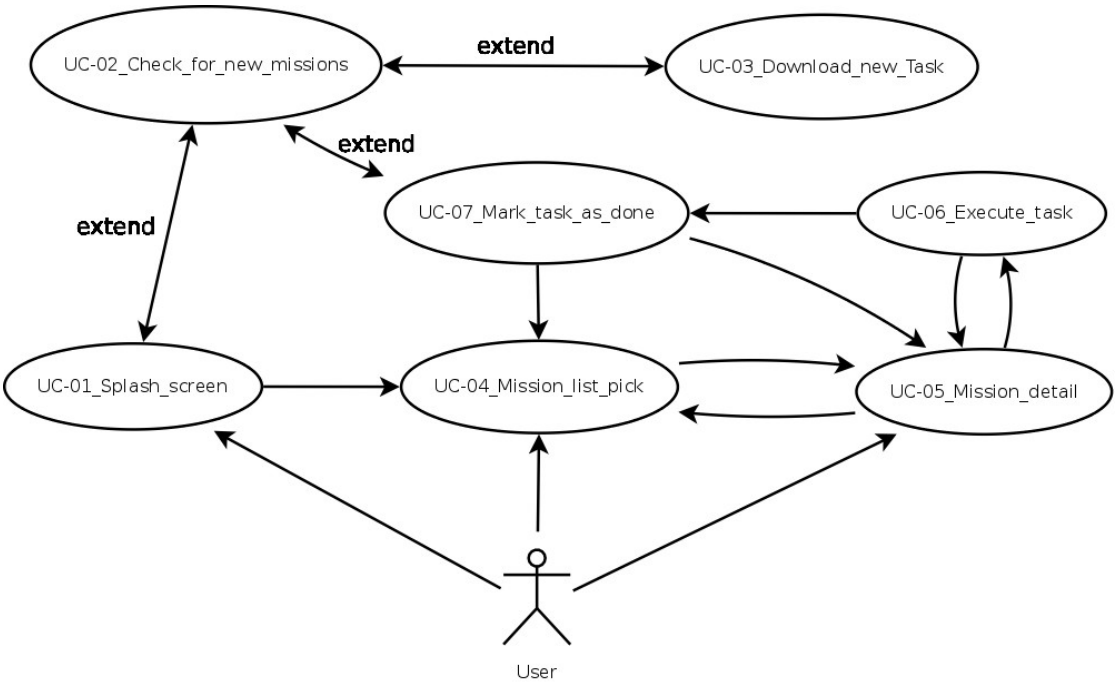

Figure 7 - Use case diagram of first pilot

Flowing table show use cases scenarios and descriptions.

UC-01_Splash_screen	
Purpose of the splash screen is to show nice graphics during load data progress, in this time also default configuration data is set if it is necessary.	
Scenario A – First run of the application	During the first run splash screen code fills the database with default data and checks possible location to contain files. The next step taken is *UC-02_Check_for_new_missions*.
Scenario B – Normal run	Wait some time to show splash for user experience purposes and go to *UC-02_Check_for_new_missions*.
UC-02_Check_for_new_missions	
This use case works like an extension and does not take over the device screen. The main task is to download new data.	
Main scenario	Send request to for available data. Compare response with database. If there is new data then save it into local database and call *UC-03_Download_new_task*, with proper arguments. *UC-04_Mission_list_pick* is next.
Alternative scenario	Device does not have a network connection, go to *UC-04_Mission_list_pick*.
UC-03_Download_new_Task	
Download full task detail.	
Main scenario	Download all data specified by the argument and save all information into database.
UC-04_Mission_list_pick	
This Activity print all missions form database also this already done.	
Main scenario	Get all relevant data from the database. When the user clicks on an available mission go to *UC-05_Mission_detail*.
UC-05_Mission_detail	
Print whole mission description, print also list of tasks to complete mission.	
Main scenario	Get from database details and print all tasks and build list of tasks. When user picks one of the task go to *UC-06_Execute_task* with proper arguments.
Alternative scenario	User goes back to mission list *UC-04_Mission_list_pick*.

UC-06_Execute_task	
Use case run correct capture activity, based on argument	
Main scenario	Based on argument start correct activity to capture external world data, just after save file on device, start service to send data. Next step is *UC-07_Mark_task_as_done.*
Alternative scenario	User presses the hardware “back button”, and cancels capture activity. Go back to *UC-05_Mission_detail*.
UC-07_Mark_task_as_done	
This part of application updates the database to set task status as done. In addition if all tasks are done, update mission status as complete	
Main scenario	Update task based on receive argument as “done”. Check statuses and set UI accordingly. *UC-04_Mission_list_pick* otherwise execute *UC-02_Check_for_new_missions* and go to *UC-05_Mission_detail*.

3.4 Transformation phase

The first pilot was not successful due to many collaborating factors, so a decision was made to completely change the form of the application to make it simpler and more transparent to the user. The Redesign included a departure from the model of big and complex missions in favor of simpler, shorter and smaller tasks. A main character was added to the game resulting in a new need in animation and sound support. Navigating the application was simplified in the main view, and only small windows of information needed to be presented. The whole send and capture modules were recycled from the old version with a list of small changes, mainly in the API interface.

3.5 Second pilot with customers

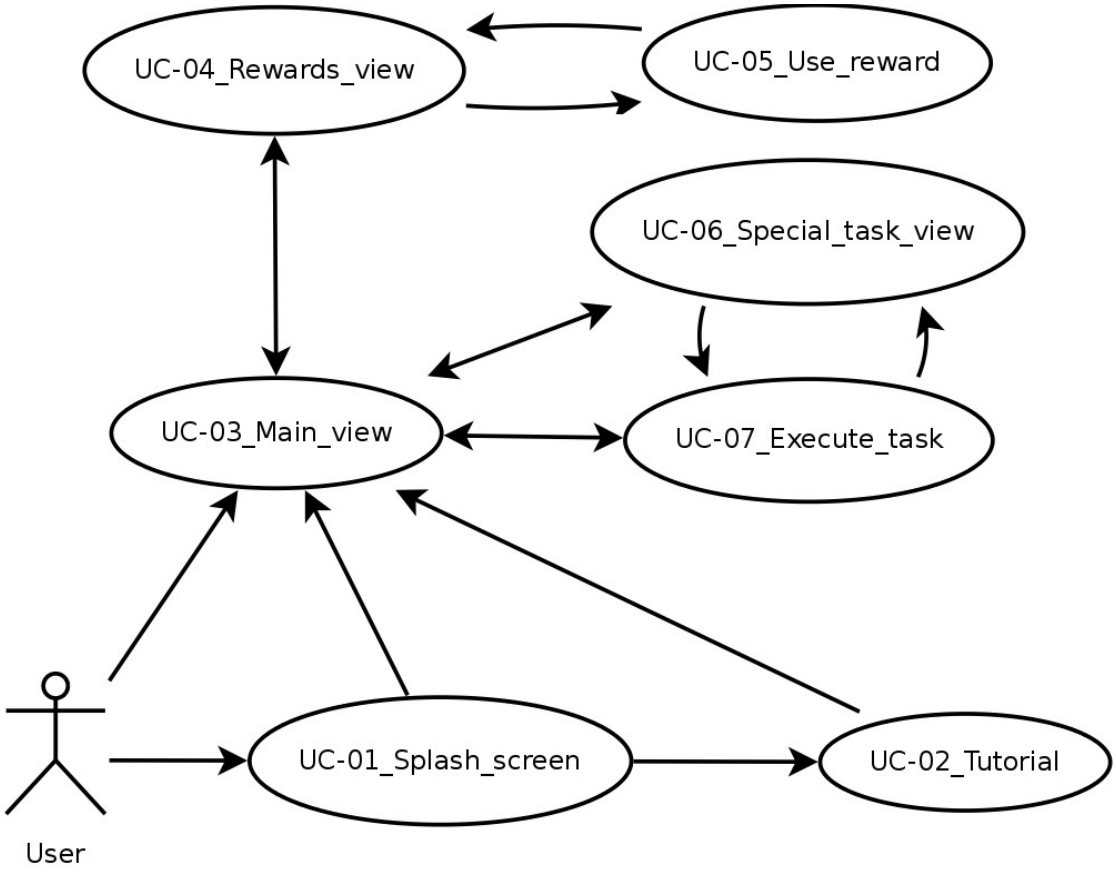

Figure 8 - Second pilot use cases diagram

Looking at figure 8 one can see immediately that the “Main view” is the center of the application. In this view the user sees character animations and can interact with them. From that place every single visible feature works as pop-over windows.

UC-01_Splash_screen	
During splash screen all libraries are loaded and a request to get data form backend is made.	
Main scenario – normal run	Show splash and wait until libraries will be ready to use. Go to UC-03_Main_view
Alternative scenario – first run	Application is running for the first time, so the next step is to introduce the user to application go to UC-02_Tutorial
UC-02_Tutorial	
The tutorial has several steps and introduces the user to the application	
Main scenario	User goes through the tutorial just reading instructions and getting acquainted to the application. After completion go to UC-03_Main_view
UC-03_Main_view	
Main view, from which the user has access to all the application's features	
Main scenario	User has access to two features; rewards and special tasks. Icons for opening this feature are hidden if user of application does not have any rewards, and does not have any special tasks to perform. From this location users have access to *UC-04_Rewards_view* and *UC-06_Special_task_view*, or also can just take new artifacts using *UC-07_Execute_task*
UC-04_Rewards_view	
This view represents all the rewards that users have achieved.	
Main scenario	Users have access to browse their own rewards with the possibility to exchange them in real live.
UC-05_Use_reward	
Use reward and exchange them for goods.	
Main scenario	Users choose reward to use and use them. In this time the application sends a request to server to mark reward as “used”. From that place users can take their reward. Applications go to *UC-03_Main_view*.

UC-06_Special_task_view	
Application offers special events. This is a place to put this event.	
Main scenario	On the screen appears a pop-over box with description of task. When user accomplish task, the meta data about task will be sent and application will return to *UC-03_Main_view*.
UC-07_Execute_task	
Main scenario	Based on the argument start correct activity to capture external world data, just after saving file on device, start service to send data. Next step is back to main screen *UC-03_Main_view*

4. Results

The module for asynchronous exchange data has been designed and implemented. The algorithm has been tested to be able to send many files at time. The module sends files one by one according to design. This part of application also has been tested to be connection loss-resistant. Transmission of data was observed to be restored when the device is online again.

The first pilot project has been designed and implemented using data exchange layer as was described above. Tests have shown that sending many files takes place in proper order using background service. Downloading data from the REST backend server works correctly and updating application content is working according to plan. Finally whole application has been tested on widest possible range of devices with positive results.

Because some of the default capture apps do not work as expected, it was decided to design and make 3 activities to capture audio, video and photos. Activities have been tested on a couple of devices with positive result. Although part of the development is successful, the application has not yet been accepted by end-users (see chapter 4.1 Result analysis).

A second pilot for testing with real end-users is still under development; however it uses the data exchange module from the first pilot, along with making small changes in the backend API specification. The logic of the second project has been redesigned

and approved by the head of the assigning company.

4.1 Result analysis

The data exchange module was adopted both in the first and second pilot with small changes. This demonstrates the design session of this layer proceeded as expected the team and headmaster. The implementation part was finished successfully, however some users for some reason did not like it. Here are some reasons that may contribute to this:

- too small advertising – couple of clients jut do not known about this application
- too complex usage – the whole application life cycle was to complex which could deter users
- network security policy - possible that the network security did not allow some users to download data

4.2 Problems during development

The biggest problem of development with mobile devices is testing on multiple devices. Right now Android works on a lot of different smart-phones made by many manufacturers. It was impossible to test product on all of these devices.

Google provides backward compatibility of the API, however core applications or libraries are a bit different between versions and do not always work exactly as older versions did. Tests were carried out on three different devices, and it turned out that the photo capture activity in one of the Android device did not return to our application screen after completing the task, resulting in the user being forced to stop the activity. When that happened, the capture activity returned an incorrect status that messaged failed operation and did not save photos at all. The devices that were used in testing had the was same manufacturer with different operating system version.

4.3 Problems during pilots

The backend REST service was designed to go perform the registration routine only once. To register the device a unique id number identifying an individual handset was used. When the developer needed to remove application and run it again, the registration procedure would fail. This resulted in return error code from the backend and no data could be downloaded from the backend. In first soution developer needed to log in into server and remove the device id manually from the backend database. When it became too burdensome server has been improved, at the next attempt to register, send the data the same as first registration, but the backend database is not changed at all.

Over time, the application grew up and took more disk space. The cause of this size bloat was new resources like images that were used in the application, applied new libraries and compiled code closed in binaries as well. Development applications need to be installed on internal memory which was not very large on one of the older devices used for development. Even if every single 3rd party application was moved to external memory, the device did not provide enough free memory space. Unfortunately, the problem can not be solved in a simple way. One option is to remove all graphics files and download them through the Internet. These files can be saved to external memory which effectively reduces the size of applications.

4.4 Suggestions for improvement

Synchronize data and network usage

Google Cloud Messaging (GCM) provides asynchronous notifications. Using this Google service, developers can send notifications to any registered device. The application installed by user can call any part of code only when a notification arrives. The first pilot used this technology, but it was dropped in the second pilot. The second project uses the most simplest way to download new data, just checking in between constant time intervals with the provided server. Initially this solution was applied

temporarily but it still use this technology. It will be much better for network usage to download new data from server only when new data appears there, instead of constantly checking for updates.

Battery life

The first pilot used a service running in the background that was any time ready for a task to execute. The Android system needs to maintain this service the whole time even if the service is waiting for new task. This results in higher hardware usage for example the CPU, resulting in shorter battery life. Service management should be rebuilt to run it only when needed and destroyed when all tasks are done.

4.5 Overall results

The module to exchange data was designed and implemented according to customer requirements and implemented in the project. Exchange layer operates successfully even in the case of a lost connection. The module can be reused easily in any other project. The use of the “Adapter” design pattern allows for quick and easy expansion module with further data exchange technology.

Development in a project in a company is completely different compared to a university project. A company project needs to be very flexible because sometimes during development new ideas appear or a new library is released. The university usually gives small problems to solve from scratch; however, during company development designers are free to use as many external libraries as wanted.

A very important ability is to design and create components that are as flexible as possible. Usually to solve small and similar university problems it is possible to redesign and rebuild an old project, on the other hand a company cannot afford to

rebuild code over and over again, for it is too time-consuming. For this reason, created parts of projects need to be flexible and easy to reuse in other places of the application.

REFERENCES

Wikipedia, Android, 2013: (Retrieved) 05.2013, http://en.wikipedia.org/wiki/Android_%28operating_system%29

Gartner, 2013: (Retrieved) 05.2013, http://www.gartner.com/newsroom/id/2335616

Wikipedia, Google Play, 2013: (Retrieved) 05.2013, http://en.wikipedia.org/wiki/Google_Play

Android developer, Android SDK, 2013: (Retrieved) 05.2013, http://developer.android.com/sdk/index.html

Android developer, Dashboards, 2013: (Retrieved) 05.2013, http://developer.android.com/about/dashboards/index.html

Android developer, App Framework, 2013: (Retrieved) 05.2013, http://developer.android.com/about/versions/index.html

Android developer, Application, 2013: (Retrieved) 05.2013, http://developer.android.com/guide/components/fundamentals.html

Android developer, Intents, 2013: (Retrieved) 05.2013, http://developer.android.com/guide/components/intents-filters.html

Android developer, Activities, 2013: (Retrieved) 05.2013, http://developer.android.com/guide/components/activities.html

Android developer, services, 2013: (Retrieved) 05.2013, http://developer.android.com/guide/components/services.html

Android developer, Broadcast receiver, 2013: (Retrieved) 05.2013, http://developer.android.com/reference/android/content/BroadcastReceiver.html

Android developer, Content providers, 2013: (Retrieved) 05.2013, http://developer.android.com/guide/topics/providers/content-providers.html

RFC 2616, 2013: (Retrieved) 05.2013, http://tools.ietf.org/html/rfc2616

REST, 2013: (Retrieved) 05.2013,
http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Wikipedia, JavaScript, 2013: (Retrieved) 05.2013,
http://en.wikipedia.org/wiki/JavaScript

Node.js, 2013: (Retrieved) 04.2013, http://nodejs.org/

Amazon AWS, 2013: (Retrieved) 04.2013, http://aws.amazon.com/s3/

Amazon pricing, 2013: (Retrieved) 05.2013, http://aws.amazon.com/s3/#pricing

Git, 2013: (Retrieved) 05.2013, http://git-scm.com/

Eclipse, 2013: (Retrieved) 05.2013, http://www.eclipse.org/